Overview of SOEP

Michael Wetter and Thierry S. Nouidui
Simulation Research Group

June 19, 2015

Lawrence Berkeley National Laboratory

SOEP overview

The purpose is to

1. understand motivation for SOEP
2. understand how SOEP is structured, and what its key modules are
   (more details will be in subsequent presentations)
3. how SOEP fits into larger eco-system of tools
4. discuss SOEP functionalities
5. discuss requirements

Motivation

Motivation

For re-modularization & encapsulation

- More flexible, testable code
- Easier, more standardized integration with other simulation tools
- Leveraging of simulation advances elsewhere, e.g., parallel solvers, system decomposition, …
- Scalability to large models
- Redeployment of models from/to other sources/use cases
  - From product specifications
  - To control systems

For re-implementation

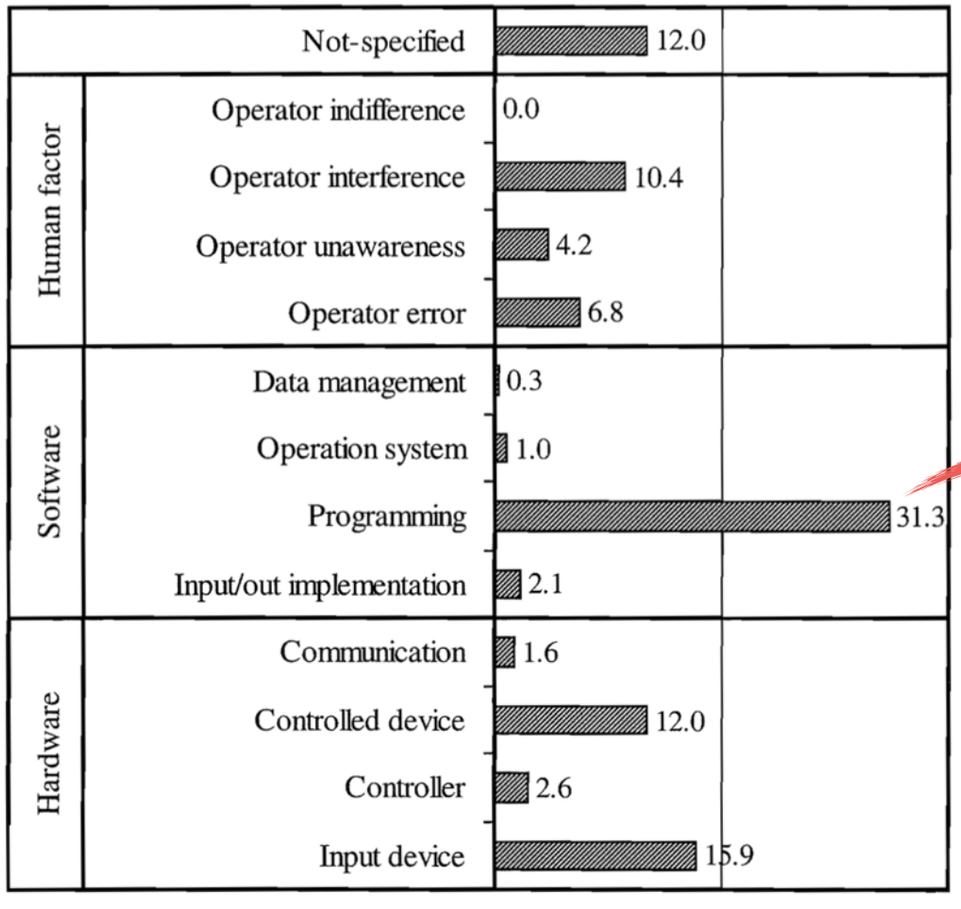
- More flexible HVAC & control
- Modeling of faults & non-idealized control
- Modeling of hybrid systems, each containing their own feedback control

Problem:
Even today's buildings don't operate as intended.
…how do we fix them *and* transition to grid-aware buildings?

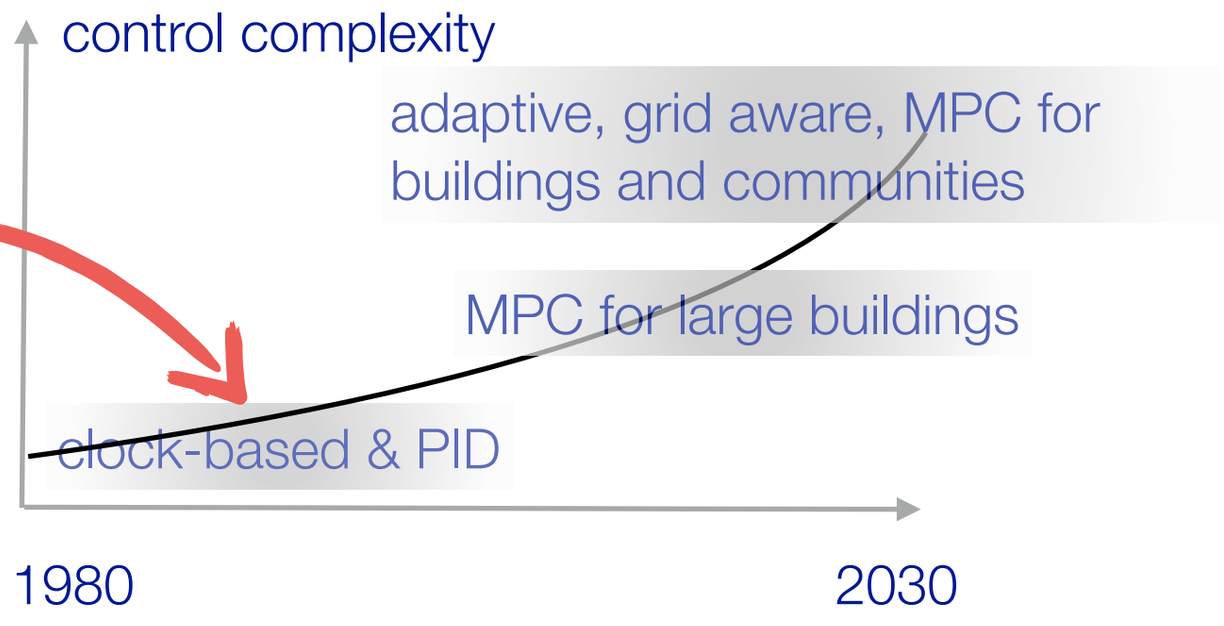
1. No means for performance quantification that caries from design to operation.

2. Building controls are broken, yet their complexity increases for ZEB.

3. Controls becomes more complex, yet there is no process that support this increased complexity.



control-related problems (Ardehali, Smith 2002)



User behavior becomes increasingly important and fixed time step simulation can cause large errors

30% difference in cooling energy for this day if 30 min vs. 1 min time steps are used.

Source:
H. Burak Gunay, William O'Brien, Ian Beausoleil-Morrison, Rhys Goldstein, Simon Breslav & Azam Khan, Coupling stochastic occupant models to building performance simulation using the discrete event system specification formalism; JBPS 7(6), 2014

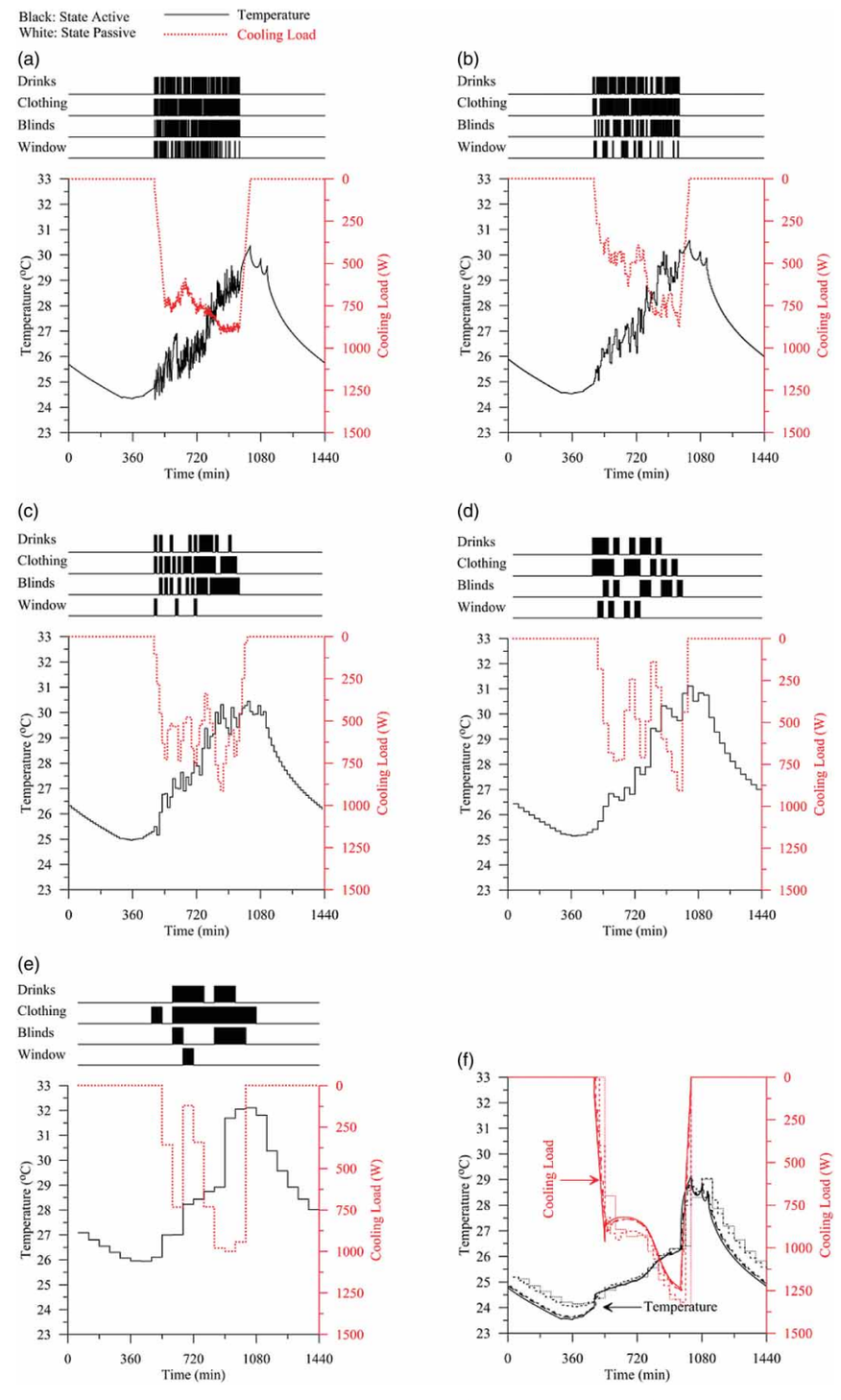
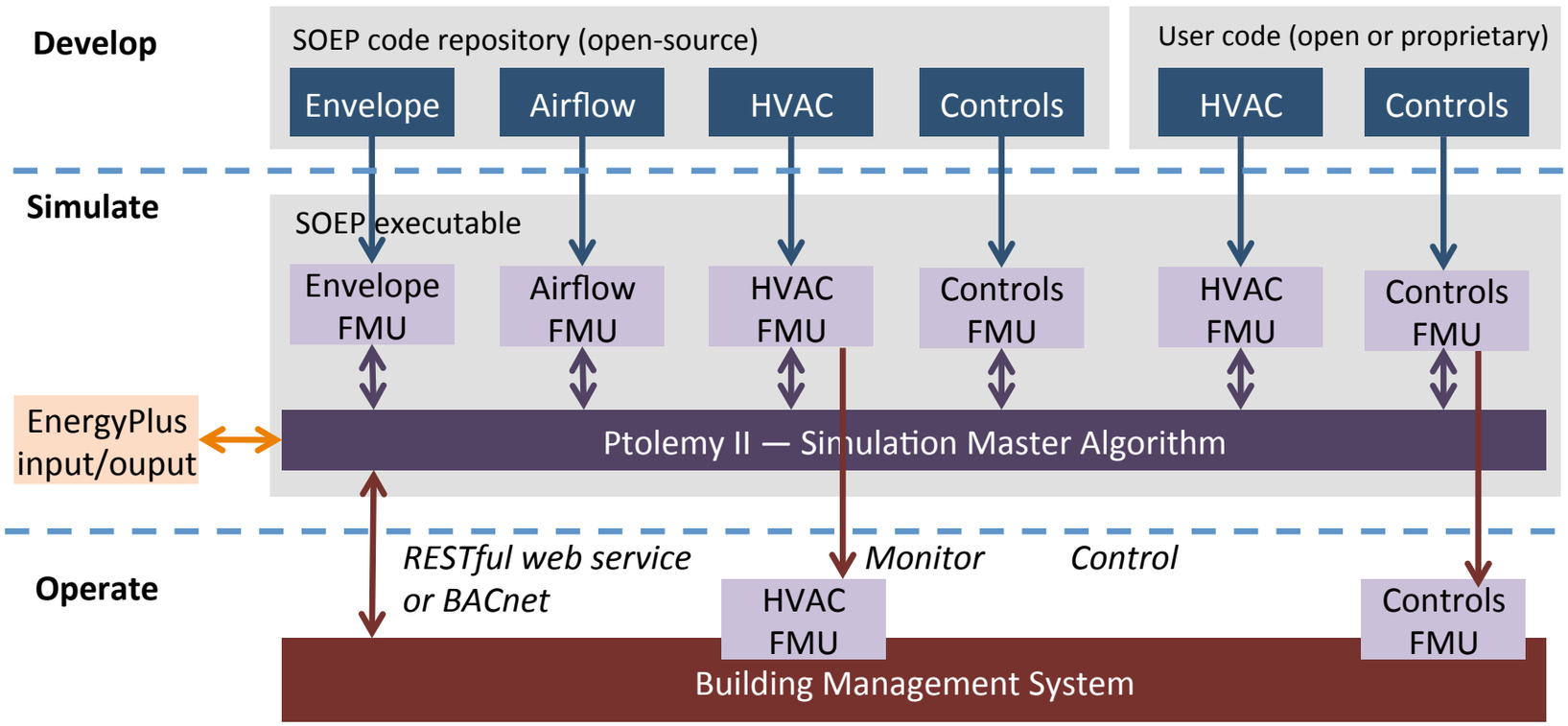


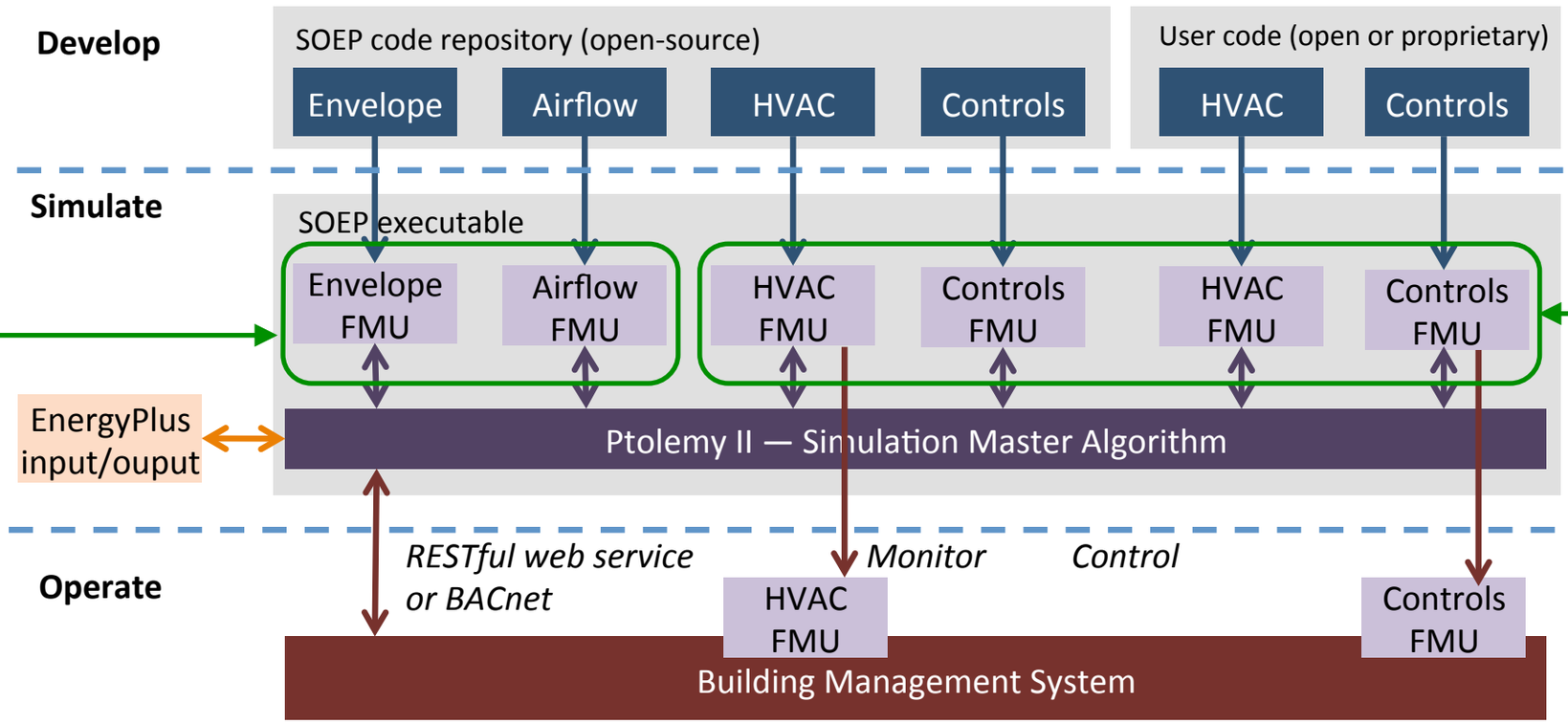
Figure 7. Operative temperature, cooling load, and adaptive states calculated with identical occupant and energy models that we simulated at time-step (a) 1 min, (b) 5 min, (c) 10 min, (d) 30 min, (e) 60 min, and (f) a reference model without an occupant model.

SOEP structure and key modules

SOEP Structure



SOEP Structure



Granularity depends on E+ redesign

To be compiled as one FMU to decrease run-time

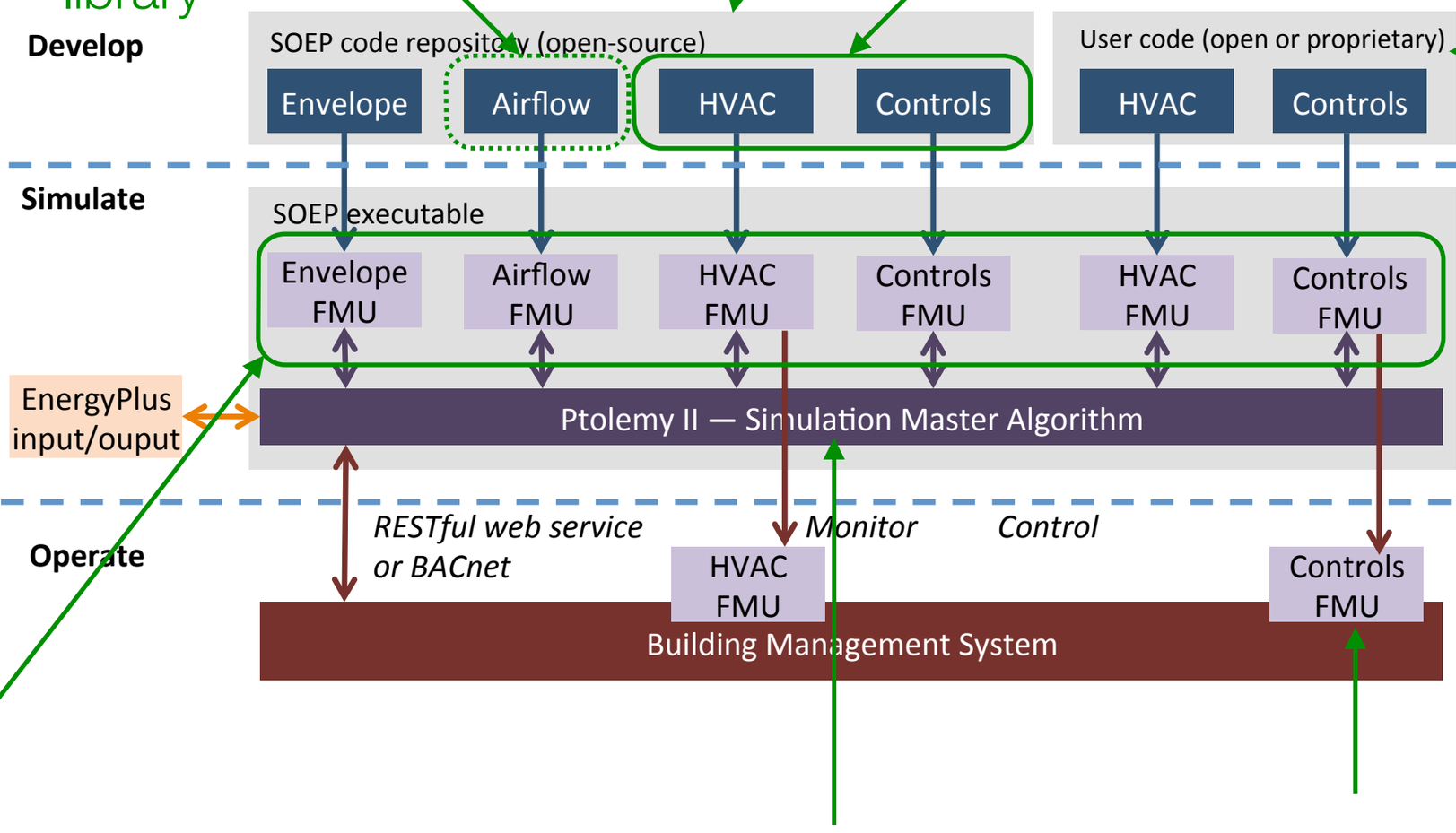
SOEP Structure

OpenStudio front-end for model instantiation and connection

Could be from Buildings.Airflow library

Modelica Buildings library

Maybe ASHRAE 205



FMUs for model-exchange, exposing differential equation (e.g., *dT/dt = f(T, t)*).

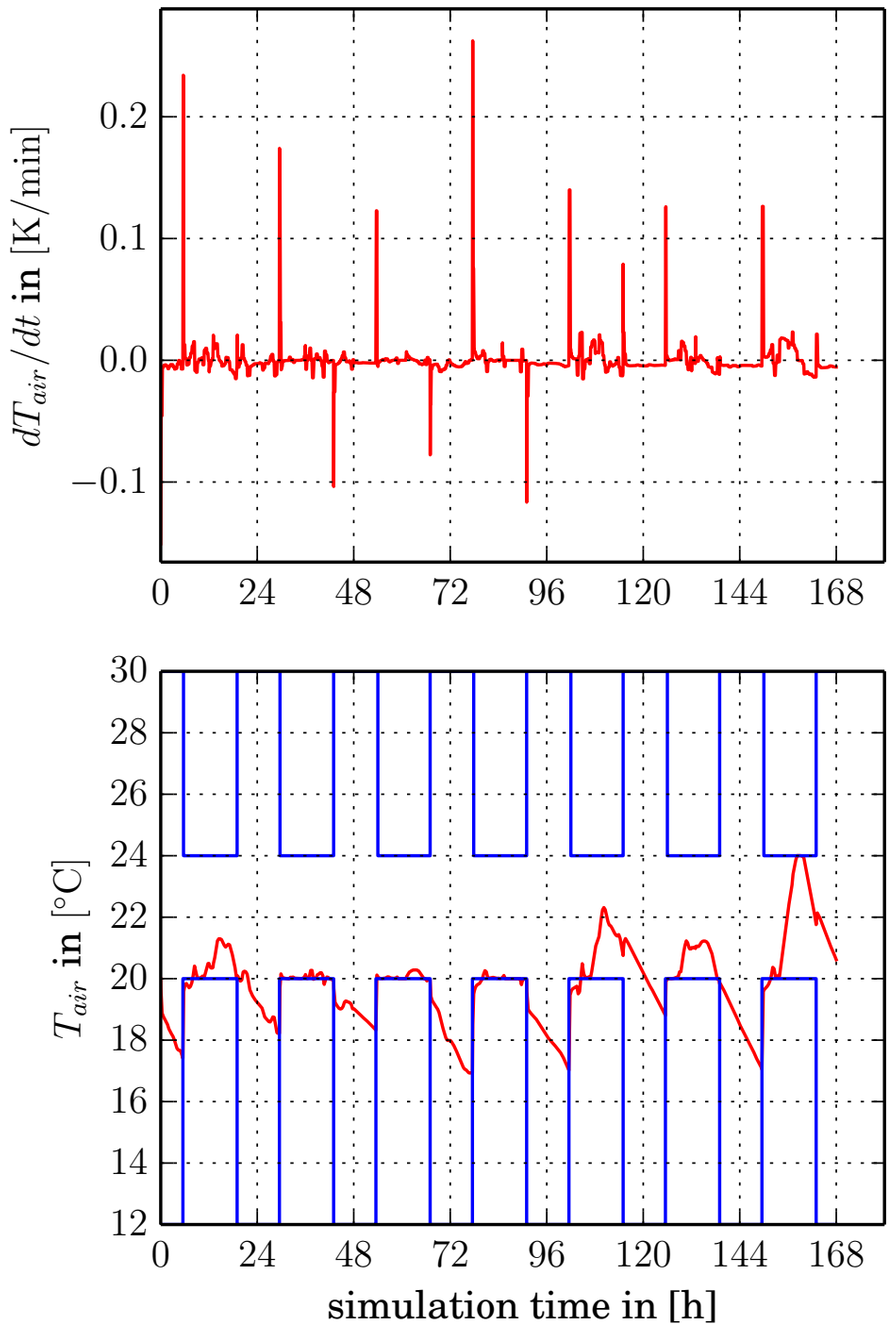
Envelope implemented by refactoring C++ code of E+

Ptolemy II provides master algorithm (discrete event simulation with QSS integration) using CyPhySim configuration

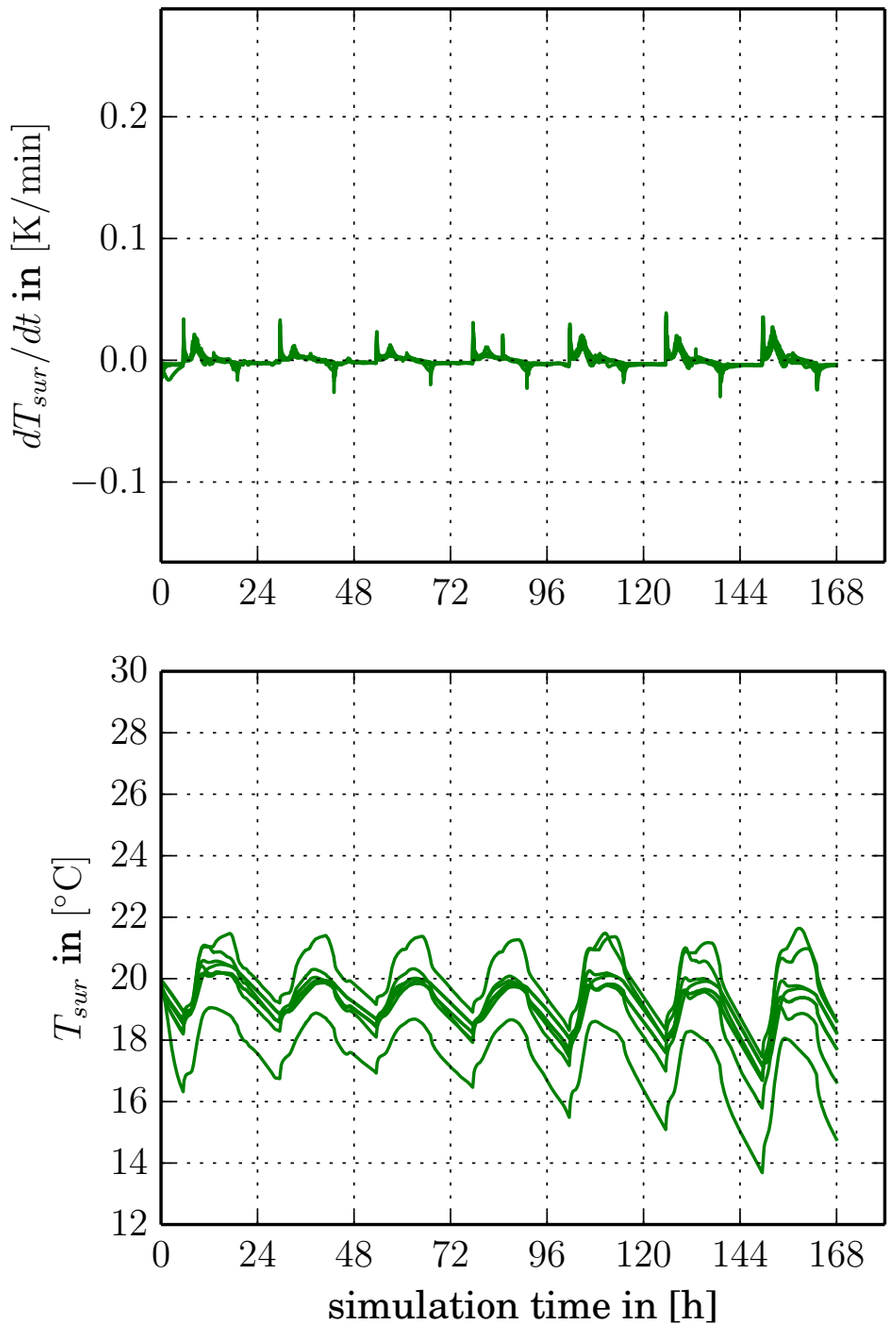
Prototyped with Tridium Niagara

Room air changes about 5 to 10 times faster than surface temperatures

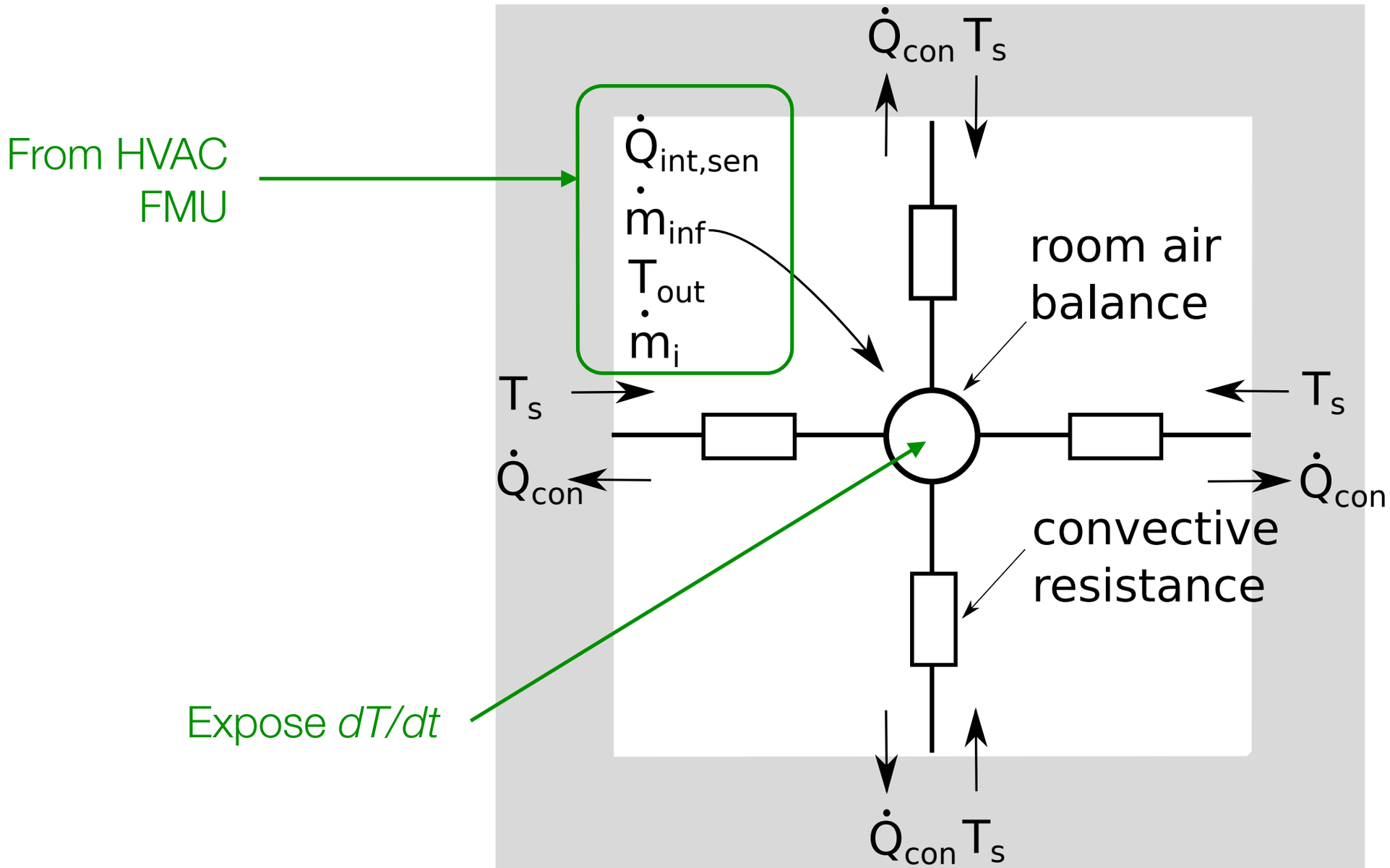
room air



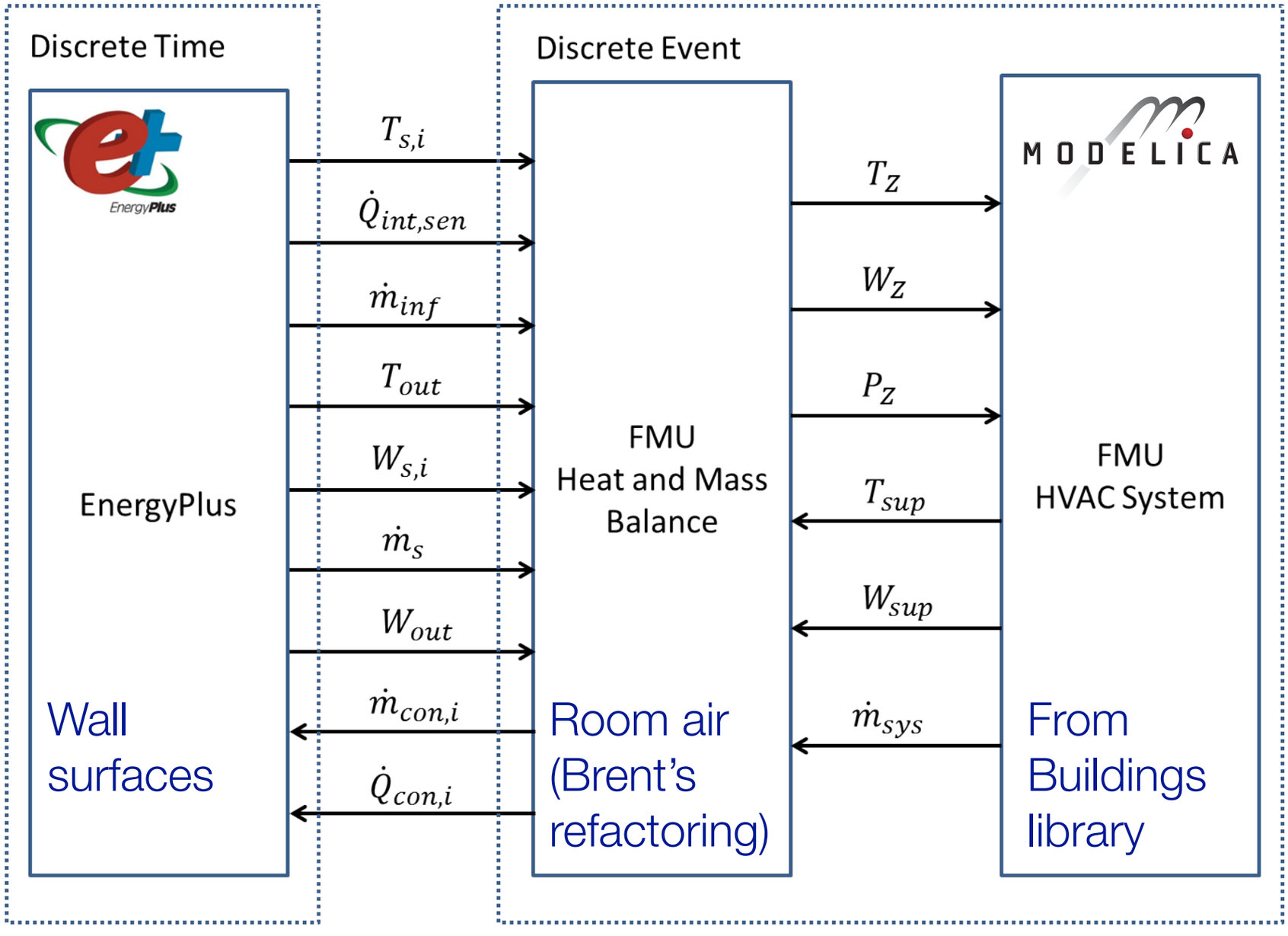
interior facing surfaces



SOEP HVAC will interface to ordinary differential equation of room air



Envelope evolves using discrete time steps while room air evolves using variable time steps (using discrete event simulation)

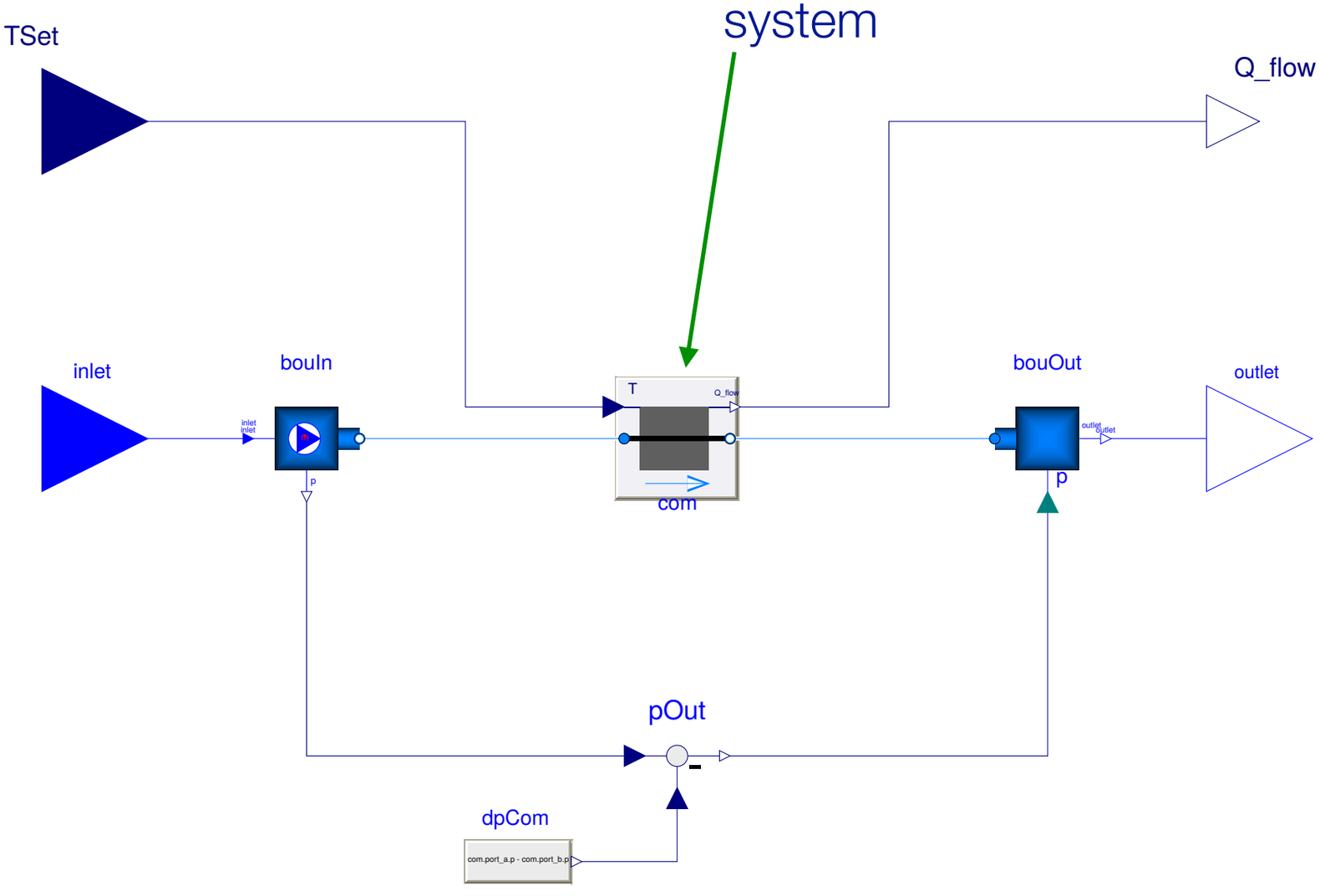


FMI container for HVAC, illustrated for an ideal heater

Interface variables for fluid connection (same for `outlet` instead of `inlet`)

```
inlet.m_flow
      p
      forward.T
              X_w
              C
      backward.T
              X_w
              C
```

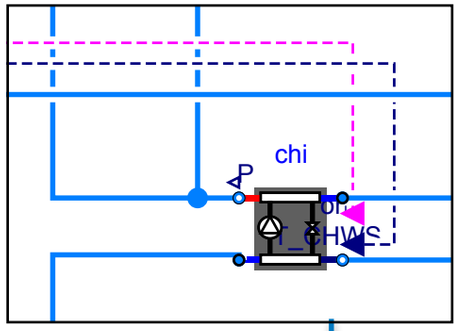
HVAC component or HVAC system

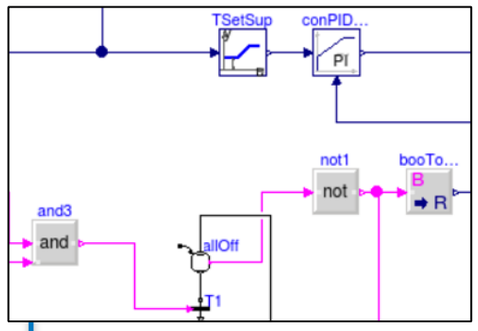


How does SOEP fit into ecosystem of other tools and activities?

Redesign EnergyPlus to allow rapid virtual prototyping, control design and model deployment for operation

HVAC & control models from open source, manufacturer libraries and ASHRAE 205





MODELICA

Develop

SOEP code repository (open-source)

Envelope | Airflow | HVAC | Controls

User code (open or proprietary)

HVAC | Controls

Simulate

SOEP executable

Envelope FMU | Airflow FMU | HVAC FMU | Controls FMU | HVAC FMU | Controls FMU

EnergyPlus input/ouput ↔ Ptolemy II — Simulation Master Algorithm

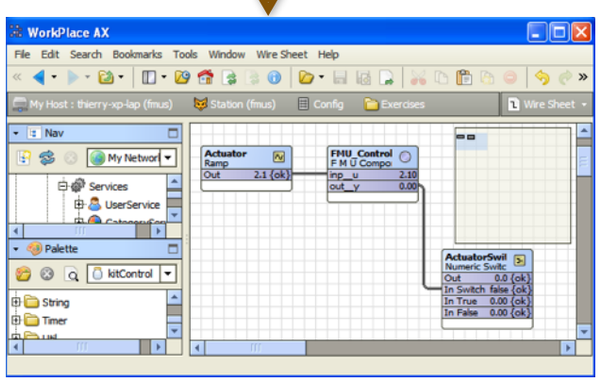
Operate

RESTful web service or BACnet — *Monitor* — *Control*

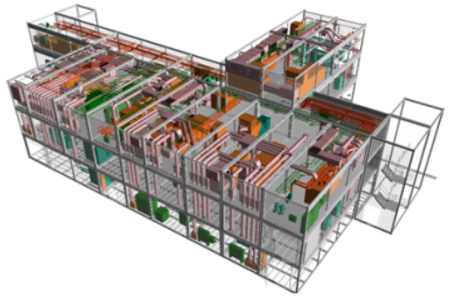
HVAC FMU | Controls FMU

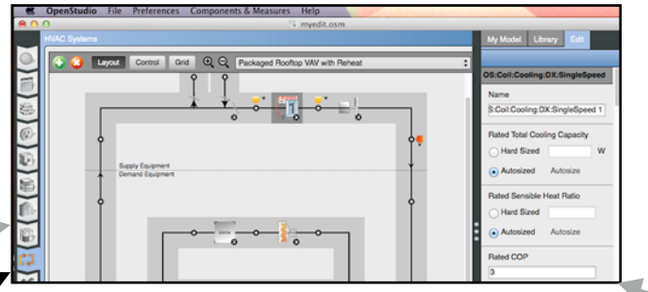
Building Management System

Control models run directly on physical controllers (e.g., Tridium)

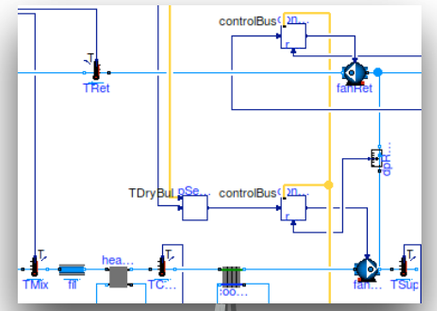


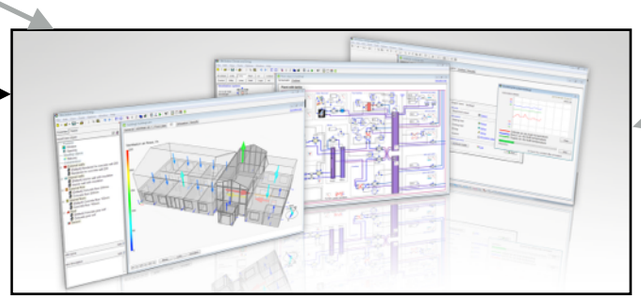
Share development of component library development (now through Annex 60)



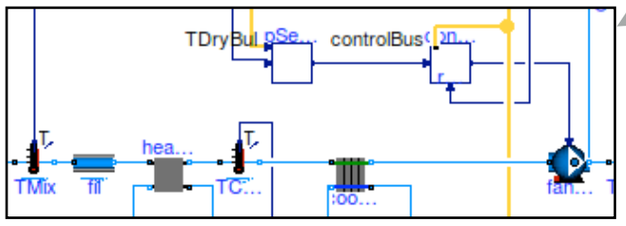


Spawn of EnergyPlus (DOE)

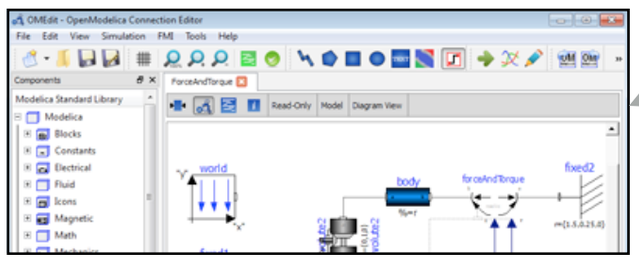




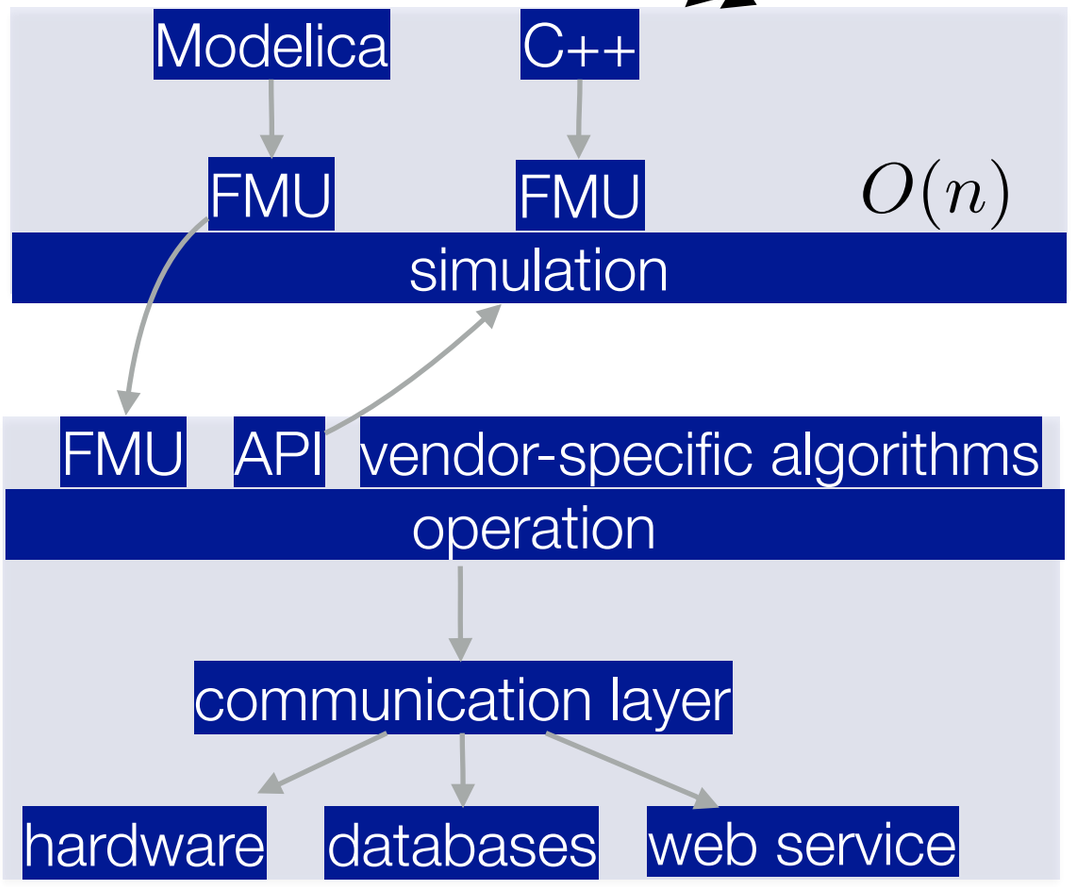
IDA/ICE (EQUA SE)



Dymola (Dassault), MapleSim, Wolfram

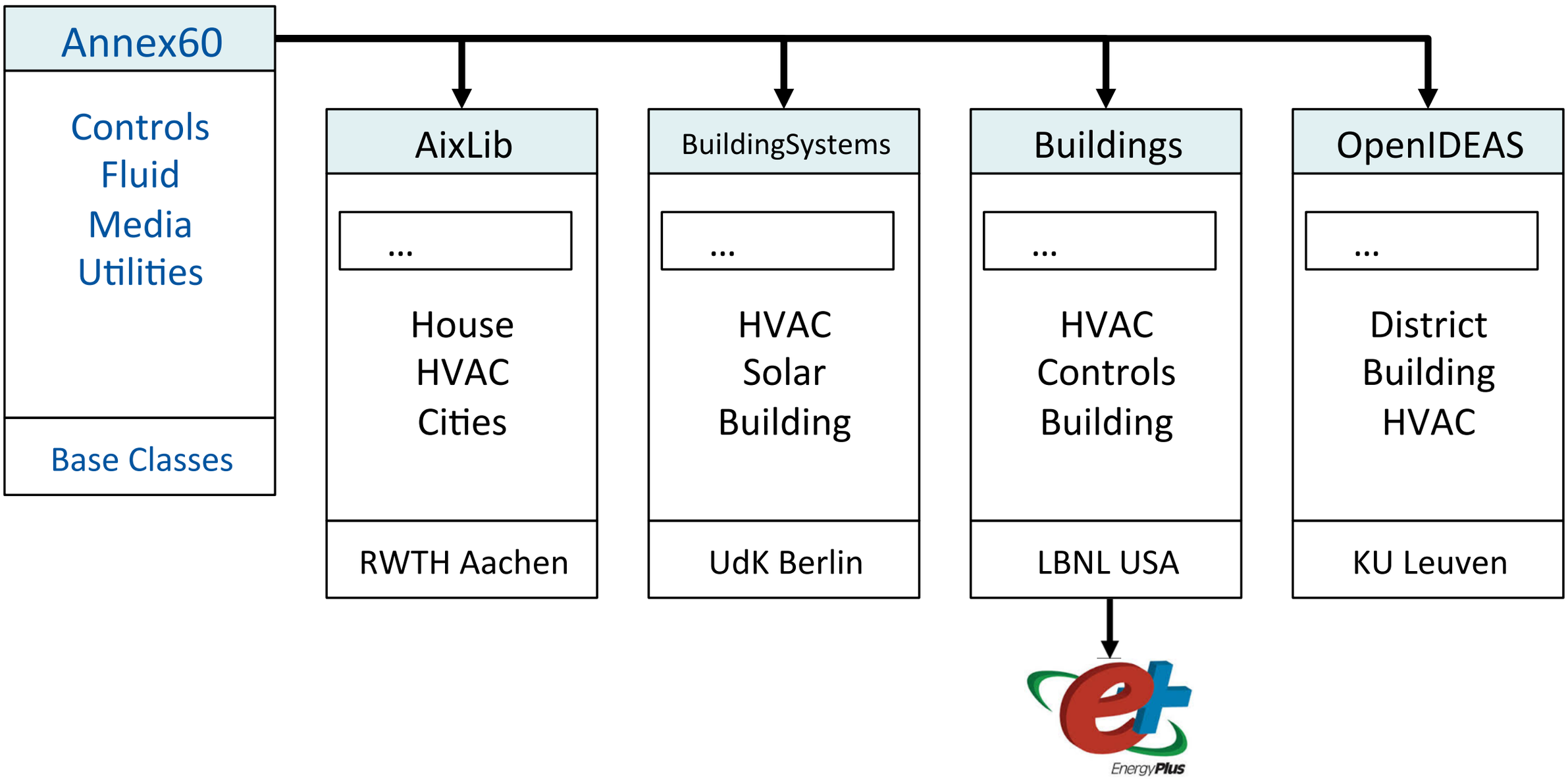


OpenModelica (Linkoeping), JModelica (Lund)



Shared Modelica HVAC library development through IEA EBC Annex 60

Goal of Annex 60, activity 1.1:
Develop and distribute a well documented, vetted and validated open-source Modelica library that serves as the core of future building simulation programs.



Functionality

Generating HVAC & building model

1. OpenStudio will generate a list of HVAC components and their connectivity, and write it to a text file.

2. OpenStudio will invoke the JModelica compiler that generates the FMU.

3. OpenStudio will generate a list of FMUs (including the one just generated) with their parameter values and their input/output connections and write xml code for Ptolemy II.

4. OpenStudio will invoke Ptolemy II FMU generation [to be discussed on Friday], and invoke EnergyPlus.

Further details and syntax in Chapter 7 of “Master Algorithm for the Spawn of EnergyPlus (Working Report)“

Algebraic Loops

Algebraic loops will be solved by JModelica.

No legacy E+ code should be inside the algebraic loop (as E+ is not differentiable).

Need at least one non-direct feedthrough in loop

allowed

$$y_2(t_2) = y_2(t_1) + \int_{t_1}^{t_2} f(u(t_1), s)\, ds$$

$$y_2(t_2) = u(t_1)$$

FMU

$$y(t_1) = u(t_1)$$

$$y_2(t_2) = y_2(t_1) + \int_{t_1}^{t_2} f(u(t_1), s)\, ds$$

$$y_2(t_2) = u(t_1)$$

Legacy E+ code

$$y(t_1) = u(t_1)$$

reject such models

$$y_2(t_2) = y_2(t_1) + \int_{t_1}^{t_2} f(u(t_1), s)\, ds$$

$$y_2(t_2) = u(t_1)$$

FMU

$$y(t_1) = u(t_1)$$

$$y_2(t_2) = y_2(t_1) + \int_{t_1}^{t_2} f(u(t_1), s)\, ds$$

$$y_2(t_2) = u(t_1)$$

Legacy E+ code

$$y(t_1) = u(t_1)$$

Autosizing

Current rule-based auto-sizing is not likely to work.

Sizing will require an iterative search.

+ Will include thermal mass effects in equipment sizing.

+ Will include control input
  (e.g., how to size a chiller if it needs to shed load during the hottest days)

- Will require multiple iterations for design day calculation.

Regression tests

Regression tests will be at three levels:

1. Modelica library unit tests
2. Ptolemy II unit tests
3. EnergyPlus unit tests

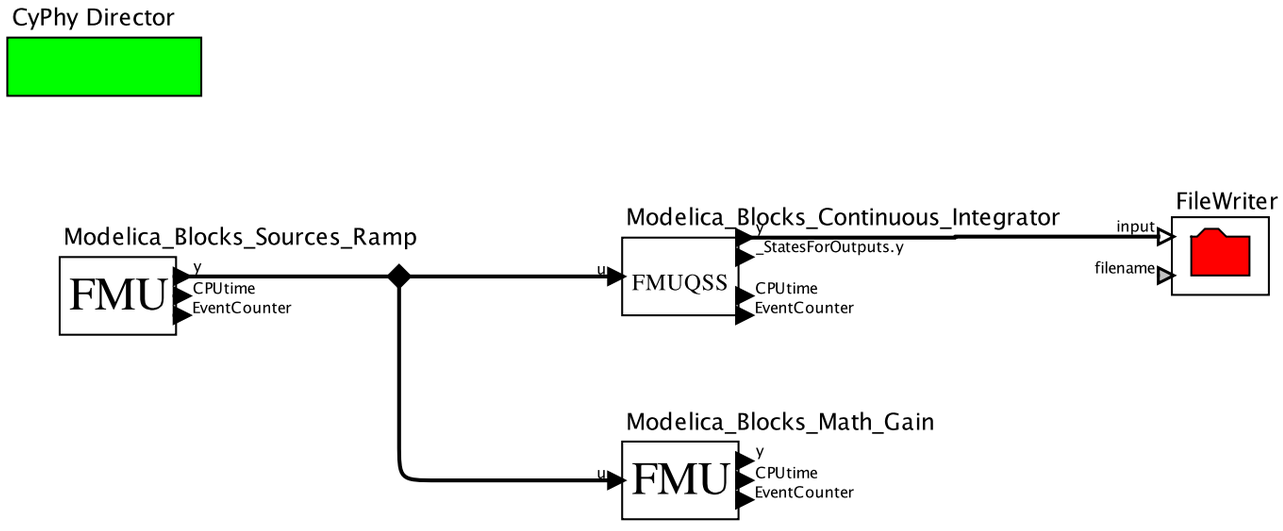
Reports for HVAC systems

The HVAC system is a discrete event simulation and hence outputs are only generated sporadically.

Using a time sample is inefficient (overhead of sampling and large files).

We recommend to store the value and — if available — derivatives to allow reconstruction of a continuous time signal.

The proposed format is on the left




```
Header
referenceValue variableName units
...
Values
referenceValue time y dy_dt d2y_dt2 ...
...
```

Further details in Chapter 7 of “Master Algorithm for the Spawn of EnergyPlus (Working Report)“

Requirements

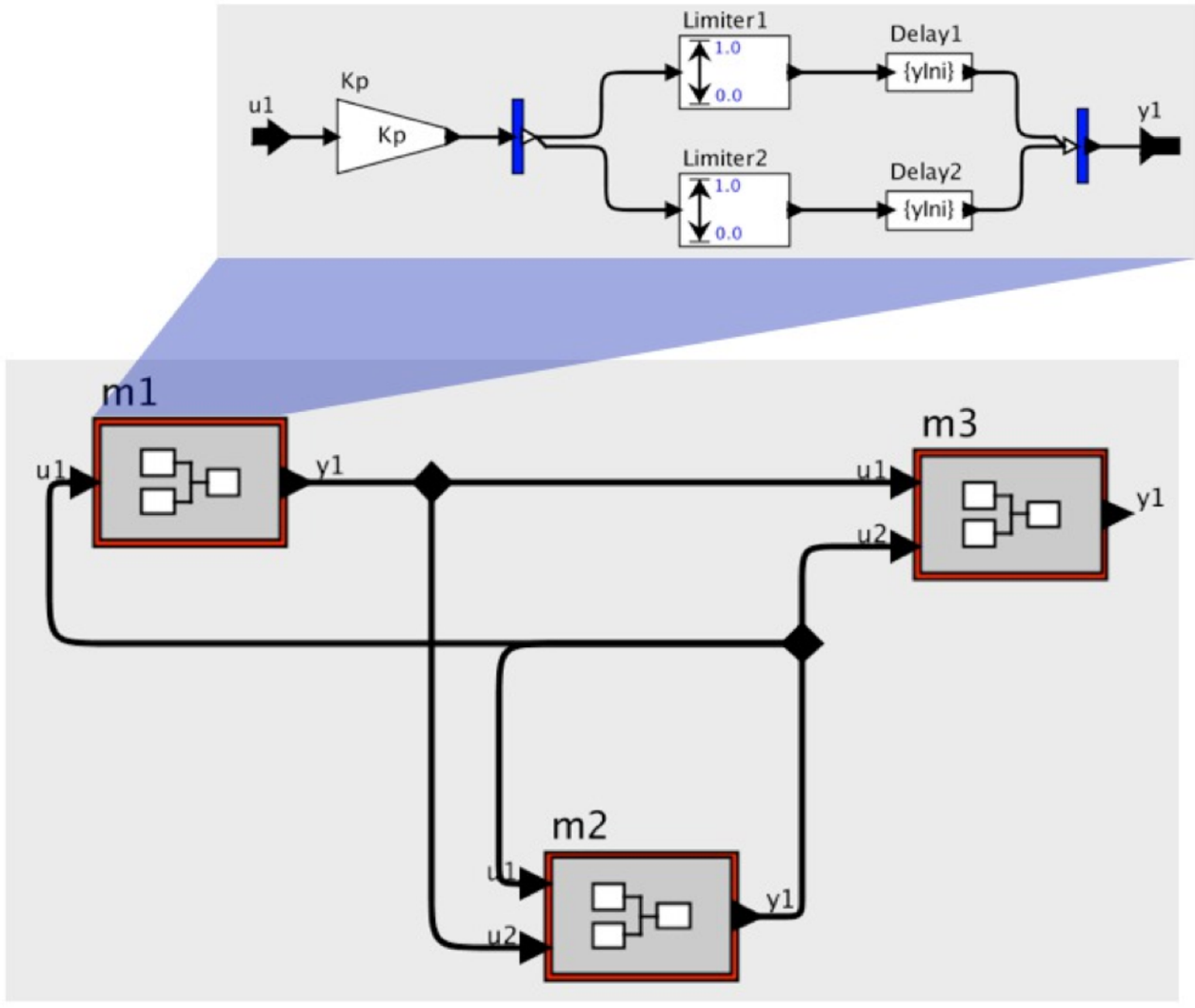
Requirements for master algorithm to build system model

List of instances with their parameter values

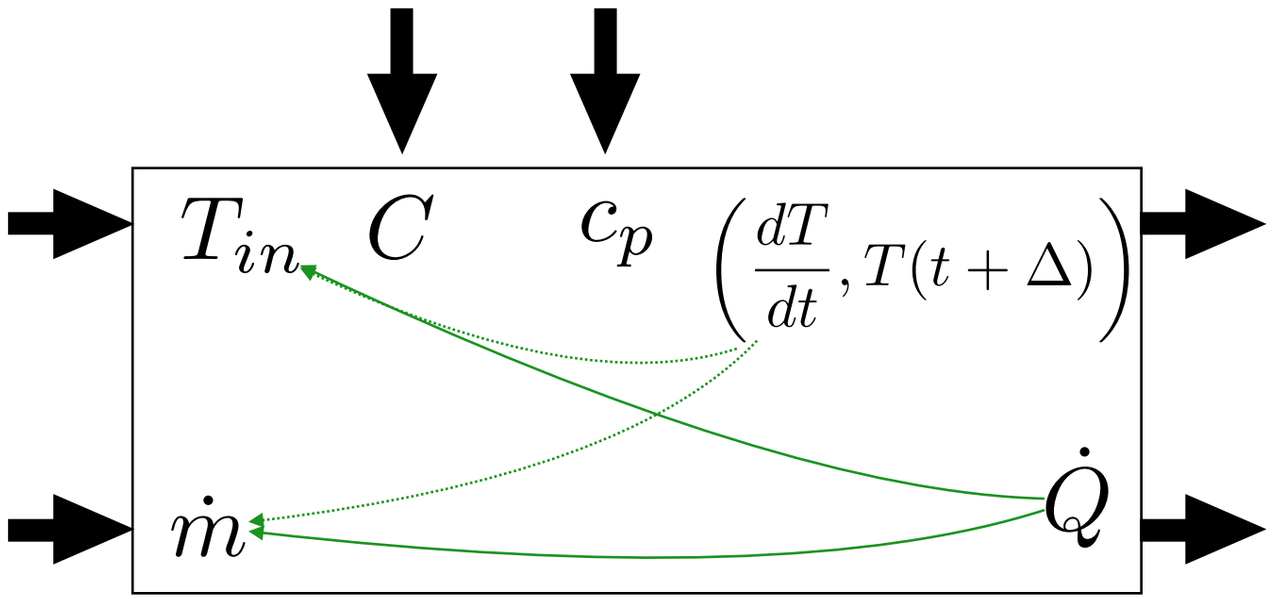
```
m1(p1=10, p2=2,
   constantGain(Kp=1)),
m2(p1=1, n=20),
m3
```

Connection list for output to input mapping

```
m1.u1 = m2.y1
m2.u1 = m2.y1
m2.u2 = m1.y1
m3.u1 = m1.y1
m3.u2 = m2.y1
```



High level requirements for computing modules (1/2)



All models need to be input/output blocks.
Optionally, they can have states.

Solver needs to be able to

- Set inputs u and states x of the module.
- Get outputs y of the module.

Model exchange (preferred):

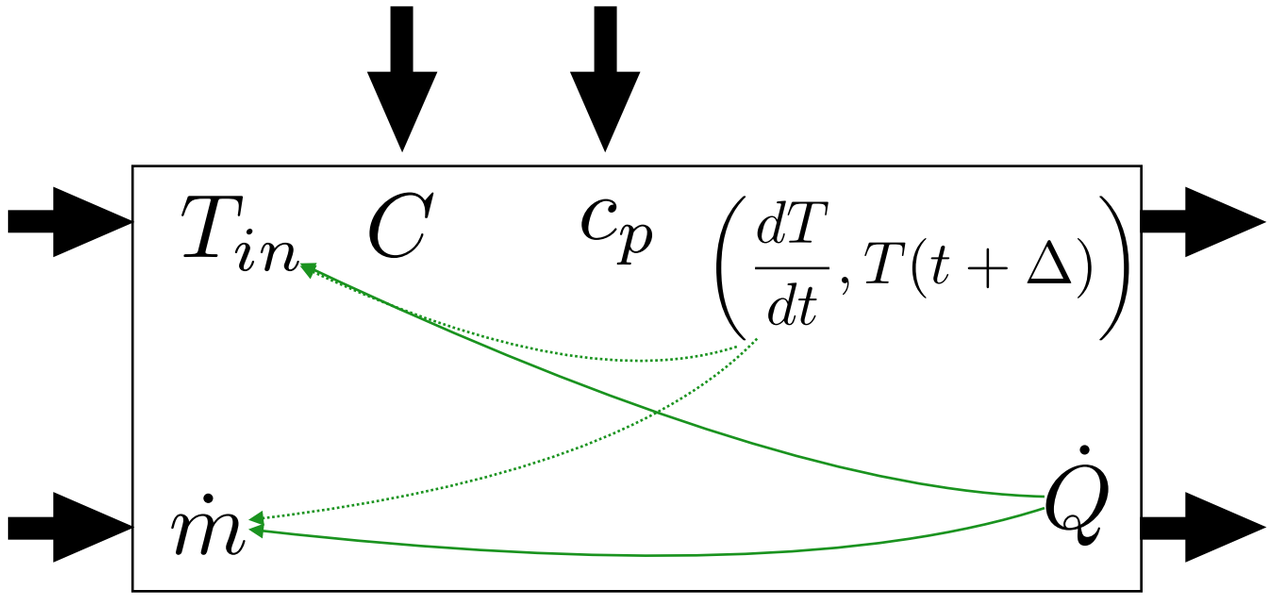
- Get time derivative *dx/dt* of the module.

Co-simulation (for E+ core):

- ask to advance time as far as the module can for the given input values $u(t_k)$.
- ask the module how far it could advance time (say to t_k+h).
- tell the module to either
  - accept states $x(t_k+h)$ or
  - redo the last step with $u(t_k)$ and $u(t_k + h')$ for some $h'<h$, and accept $x(t_k + h')$.

High level requirements for computing modules (2/2)

Fluid flow models must have the semantics of the stream connector. This need is orthogonal to our implementation, but needed for robustness if airflow networks are coupled with feedback control and thermal models.



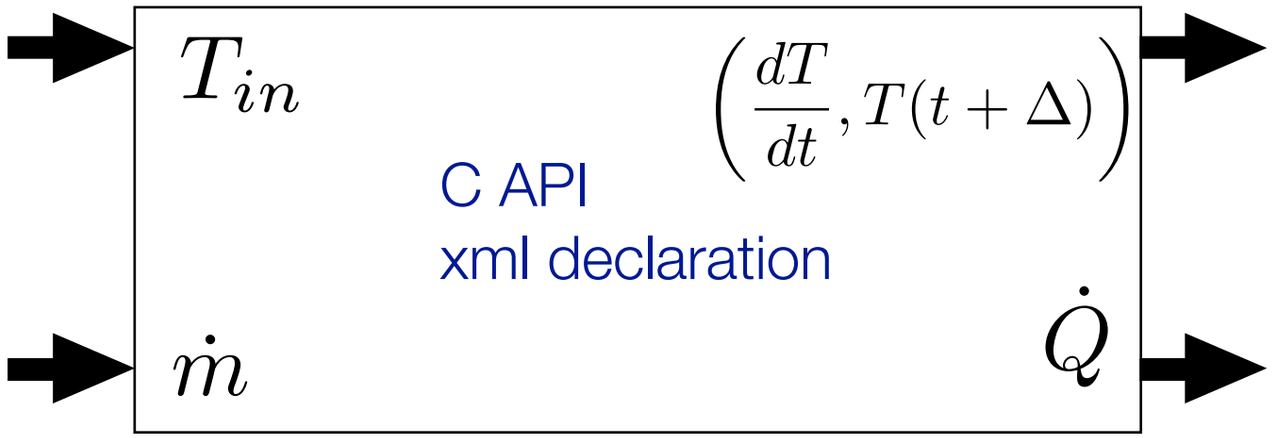
For QSS efficiency, provide what outputs and state derivatives directly depend on inputs.

Incidence matrices

- what output depends directly on what input
- what state derivative depends on what input and state
- scaling for all variables (e.g., 100,000 Pascals vs. 0.01 kg/s flow rate)

For computational efficiency, like to only evaluate equations whose right-hand side changed.

Transition of EnergyPlus engine



Stage 1:

Convert discrete time simulators of thermal zones to a continuous time simulator (from co-simulation to model exchange). This continuous time model is then coupled to QSS.

Stage 2:

Link E+ envelope to QSS based models for HVAC, control and multi zone airflow.

Expose E+ envelope so that it satisfies module requirements, i.e., an input/output block, optional with memory.

From OpenStudio, generate model description file for coupling.

Refactor E+ envelope outputs that have direct feedthrough (e.g., that depend algebraically on the input) to be differentiable whenever they may become part of an algebraic loop — or alternatively, ensure no such direct feedthrough exists.

Stage 3:

Break core up into individual thermal zones, each being an input/output block. Otherwise, there will be a substantial performance loss as the sparsity of QSS is not exploited.

Questions and discussions

1. SOEP structure
2. larger eco-system of tools
3. functionalities
4. requirements